Hardware Design

**Lab 3 Report**

Sequential Circuits

**Team 01**

112062122 謝佳晉 112062144 范升維

Table of Contents:

[Advanced Q1. 4-bit Ping-Pong Counter 1](#_Toc179474120)

[Advanced Q2. First-In First Out (FIFO) Queue 4](#_Toc179474121)

[Advanced Q3. Multi-Bank Memory 6](#_Toc179474122)

[Advanced Q4. Round-Robin FIFO Arbiter 9](#_Toc179474123)

[Advanced Q5. Exhaustive testbench design 10](#_Toc179474124)

[FPGA: 4‑bit Parameterized Ping‑Pong Counter 10](#_Toc179474125)

[Testbenches 11](#_Toc179474126)

[A. 4-bit Ping-Pong Counter 11](#_Toc179474127)

[B. First-In First Out (FIFO) Queue 11](#_Toc179474128)

[C. Multi-Bank Memory 11](#_Toc179474129)

[D. Round-Robin FIFO Arbiter 11](#_Toc179474130)

[E. 4-bit Parameterized Ping-Pong Counter 11](#_Toc179474131)

[What have we learned from Lab 3? 11](#_Toc179474132)

[Contributions 11](#_Toc179474133)

# 一張含有 螢幕擷取畫面, 正方形, 對稱, 黑色 的圖片 自動產生的描述一張含有 螢幕擷取畫面, 設計 的圖片 自動產生的描述Advanced Q1. 4-bit Ping-Pong Counter

* 3 inputs: *clk, rst\_n, enable*
* 2 outputs: *direction, out[3:0]*

一張含有 黑色, 螢幕擷取畫面, 黑暗 的圖片

自動產生的描述In this question, we introduce an Up-down counter to implement the function of changing counting direction.

(Upper-output)

(Lower-output)

The leftmost part of the Up-down counter is the **Direction control**. Input-Output is shown in the table below. (input, output)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Up | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Down | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| Enable | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Lower-output | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1  (won’t happen) |
| Upper-output | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1  (won’t happen) |

The values of Up and Down can be traced back to Figure 1.1. We get their Boolean expressions by **drawing K-map** on **rst\_n, direction, and border**(out==15 or out==0).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Direction | border | rst\_n | Up | Down |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

The Boolean expressions is shown below:

Up = rst\_n & (( !direction & border ) | ( direction & border )) = rst\_n & ( direction ^ border )

Down = rst\_n & (( direction & border ) | ( !direction & !border ))

For the value of direction, we use JK flip flop. Similarly, we **draw K-map** on **rst\_n and border**.

|  |  |  |  |
| --- | --- | --- | --- |
| Border | rst\_n | J of JKFF | K of JKFF |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

The Boolean expressions is shown below:

J of JKFF = !rst\_n | border.

K of JKFF = rst\_n & border.

For *out[3:0]*, because it will be set to *4’d0* if *rst\_n* == 0, a MUX takes on *rst\_n* before TFF input is needed. We want each bit of *out[3:0]* to be *0*, but we can’t just simply input *0* into the MUX because of the feature of TFF(Toggle when input *1*, Nothing change when input *0*). This is where another MUX taking on each bit of *out[3:0]* comes in.

一張含有 文字, 螢幕擷取畫面, Rectangle, 黑色 的圖片

自動產生的描述

If the bit is *1’b1*, *1’b1* will be set to output in order to toggle the TFF value. If the bit is *1’b0*, *1’b0* will be set to output, and the TFF keep its value. In this way, every bit will all be *1’b0* when *rst\_n == 0*.

# Advanced Q2. First-In First Out (FIFO) Queue

一張含有 螢幕擷取畫面, 時鐘, 設計 的圖片

自動產生的描述

▲Figure 2. 2 overall circuit for error

* 3 inputs: *clk, rst\_n, din[7:0]*
* 2 outputs: *dout[7:0], error*
* 3 internal signals: *cnt[3:0], Raddr[2:0], Waddr[2:0]*

To construct a memory unit, we combine two MUXes and a DFF. At first, we use rst\_n as selection, if rst\_n == 0,

一張含有 螢幕擷取畫面, 正方形, Rectangle, 設計 的圖片

自動產生的描述

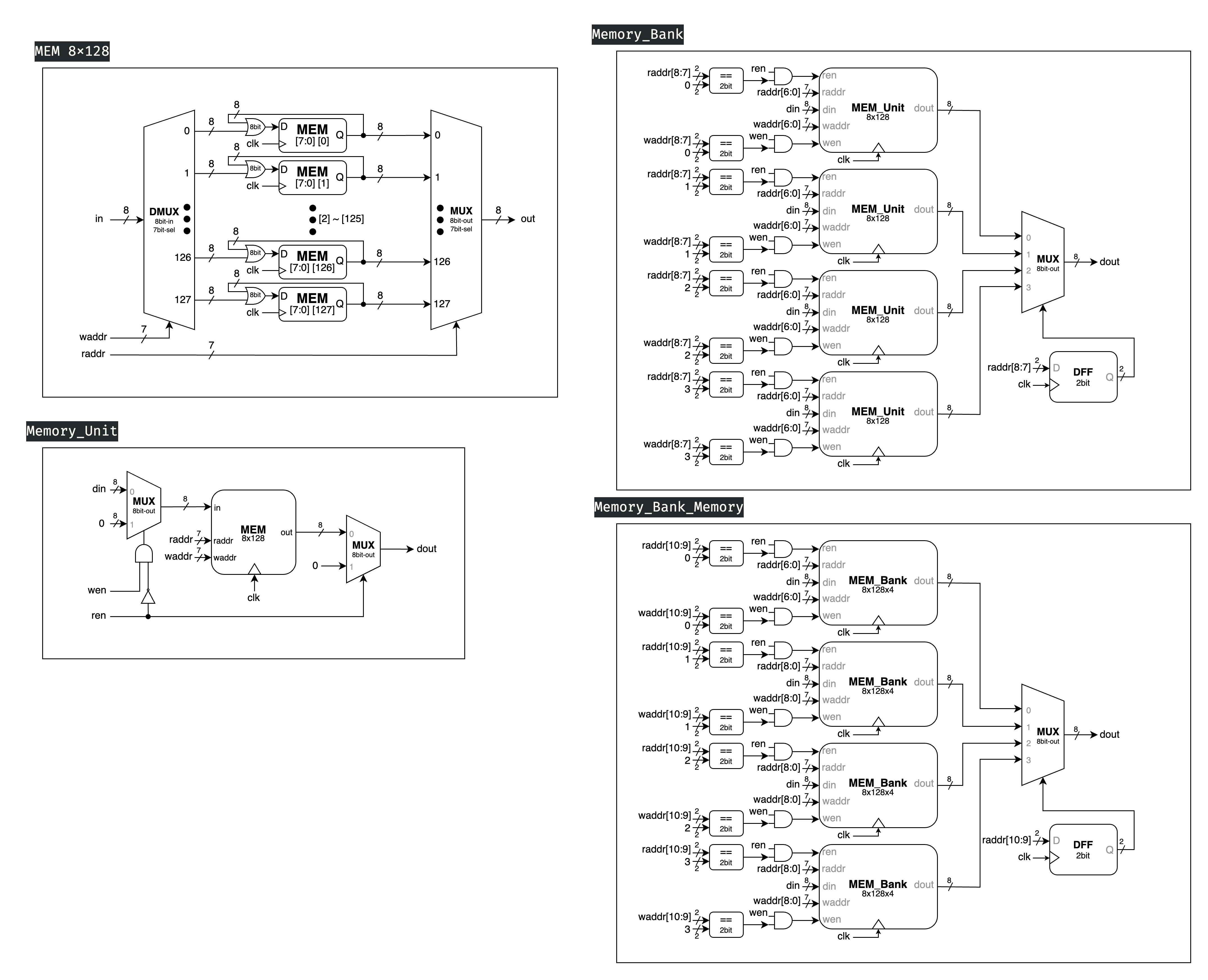
▲Figure 2. 3 overall circuit for cnt

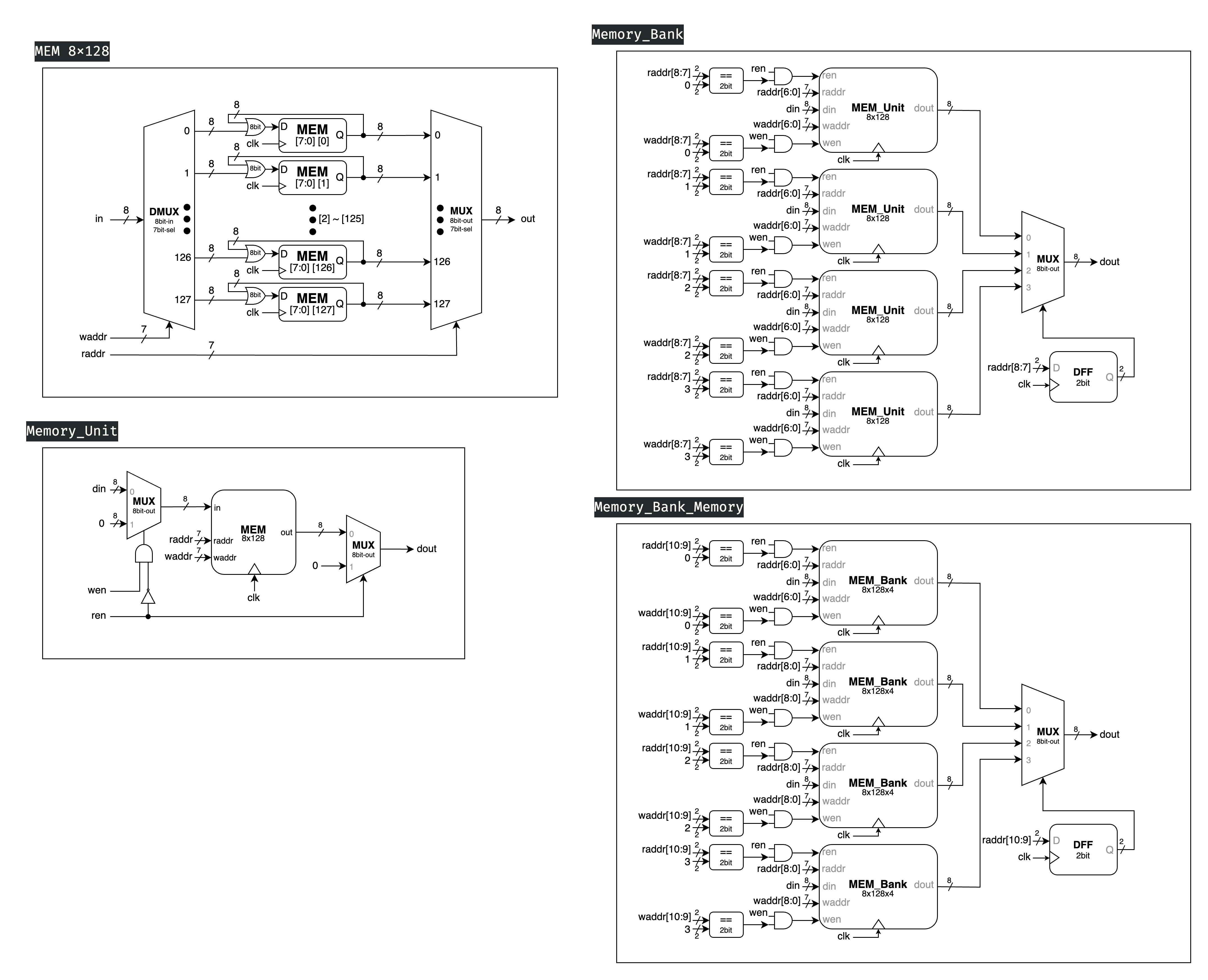
一張含有 螢幕擷取畫面, 文字, Rectangle, 正方形 的圖片

自動產生的描述

▲Figure 2. 4 overall circuit for Raddr

# Advanced Q3. Multi-Bank Memory

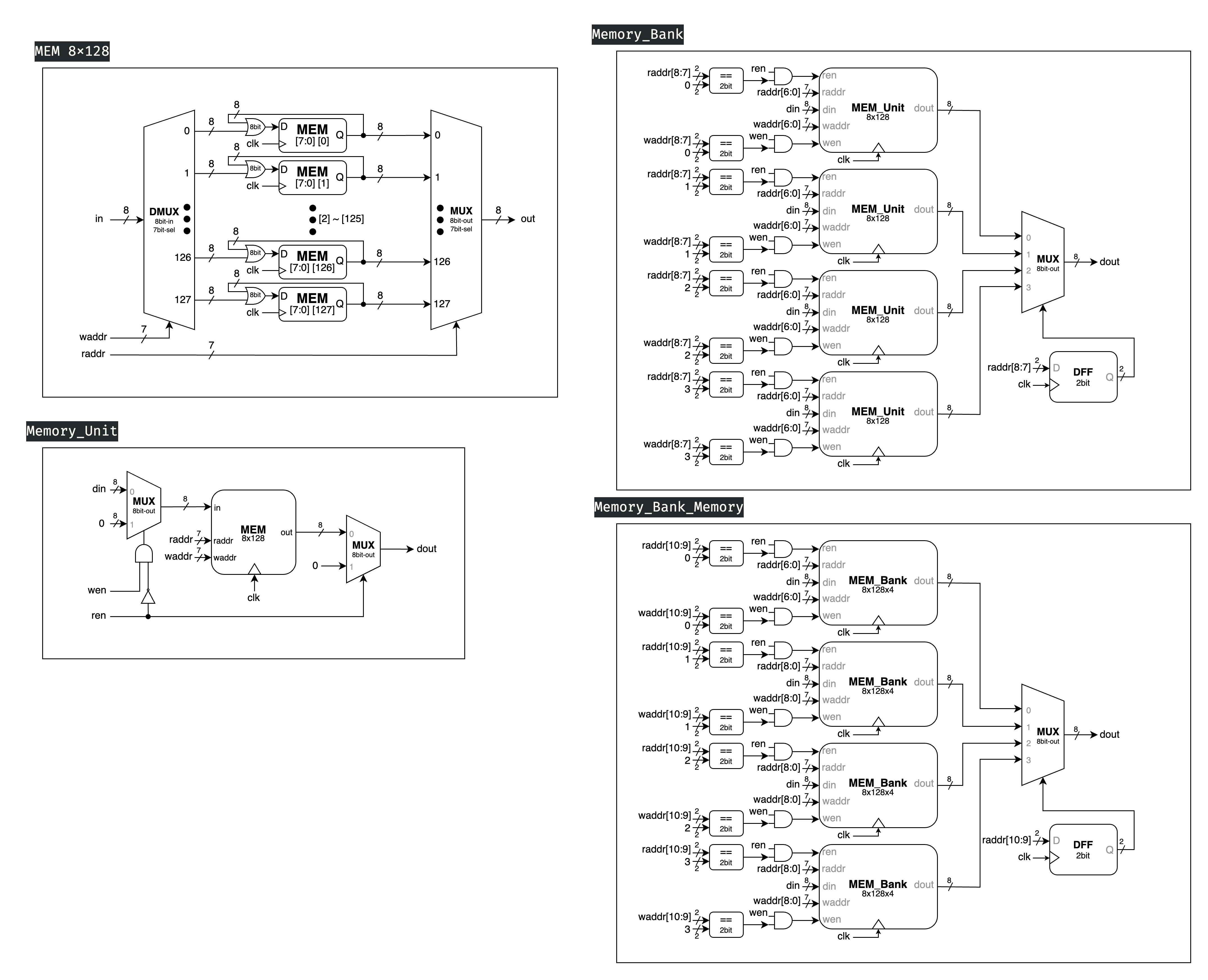
In Advanced Question 3, we are required to construct a multi-bank memory system consisting of 4 **Memory Banks**. Each bank comprises 4 **Memory Units**, and each Memory Unit is an 8-bit x 128 memory module, which we previously implemented in Basic Question 2.

▲ Figure 3.1: MEM 8x128

▲ Figure 3.2: Memory Unit (8bit x 128 memory module)

* 6 inputs: *clk, ren, wen, waddr[6:0], raddr[6:0], din[7:0]*
* 1 output: *dout[7:1]*

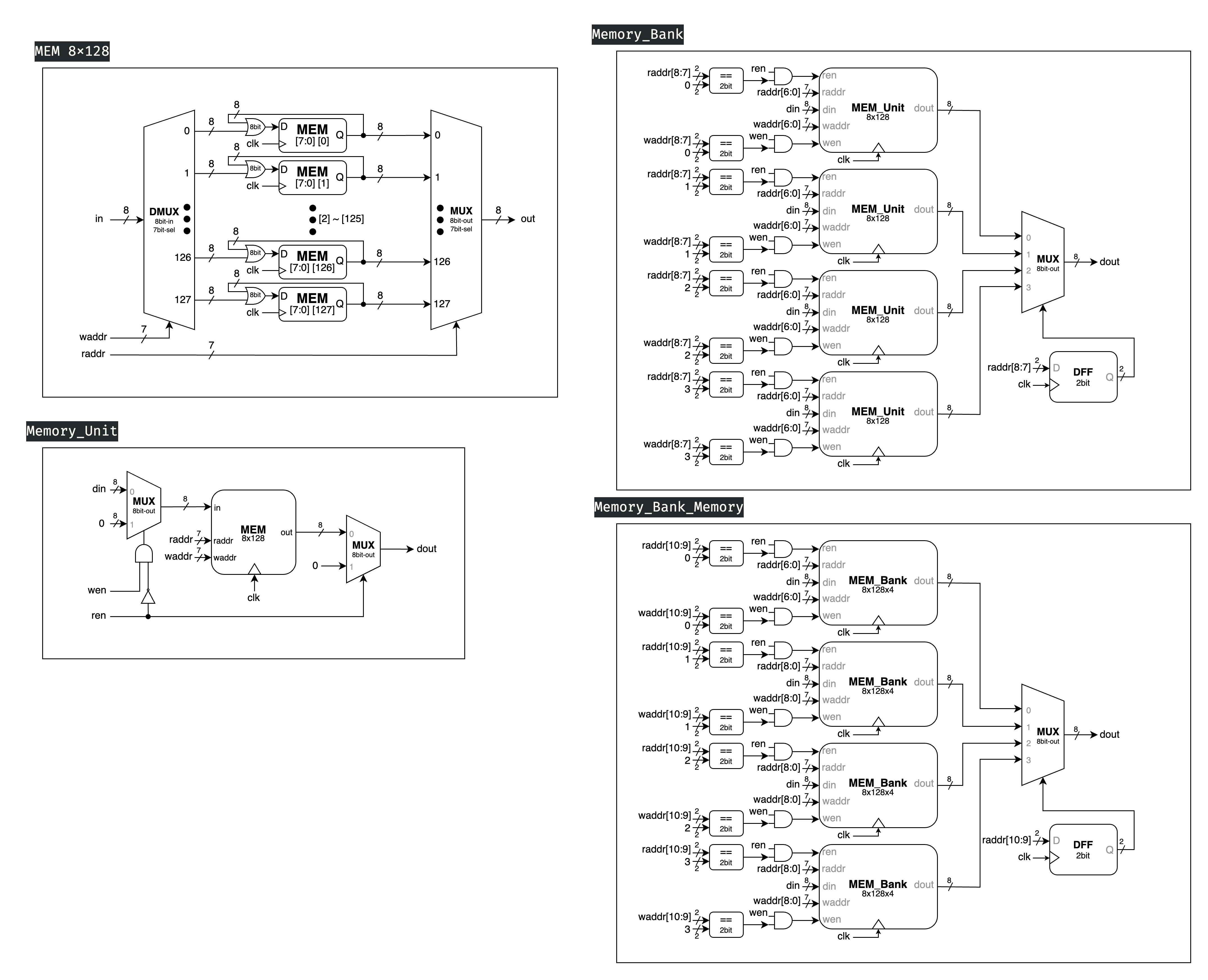
Figure 3.2 illustrates our implemented **Memory Unit**, which is an 8-bit x 128 memory module. This unit incorporates the **MEM 8x128** component shown in Figure 3.1. The MEM 8x128 is designed to store the input data (*din*) at the memory location specified by the write address (*waddr*). Each memory location within the MEM consists of a synchronized 8-bit register array, implemented with D Flip Flop. The output (*dout*) is determined by the value stored in the memory location designated by the read address (*raddr*).

Figure 3.2 illustrates the implementation that fulfills the specified requirements of the problem. The circuit is configured to output *dout[7:0]* only when the read enable signal (*ren*) is asserted to *1'b1*; otherwise, it outputs *8'b0*. Furthermore, the circuit permits data input operations under the condition that the input data (*din*) is stored in the memory location designated by the write address (*waddr*) exclusively when the write enable signal (*wen*) is asserted to *1'b1* and, concurrently, the read enable signal (*ren*) is de-asserted to *1'b0*. This control scheme ensures proper read and write operations while preventing potential conflicts between simultaneous read and write attempts.

▲ Figure 3.3: Memory Bank

* 6 inputs: *clk, ren, wen, waddr[8:0], raddr[8:0], din[7:0]*
* 1 output: *dout[7:1]*

Figure 3.3 depicts the circuit diagram of the **Memory Bank**, which integrates four Memory Units. The *waddr* and *raddr* signals have been expanded by two bits to specify the target Memory Unit. Consequently, the write enable (*wen*) and read enable (*ren*) signals for each Memory Unit are activated (set to *1'b1*) only when this module's *wen*/*ren* is *1'b1* and the two most significant bits of *waddr*/*raddr* correspond to that particular Memory Unit.

The output (*dout*) of this module utilizes a multiplexer (**MUX**) to select the *dout* from the Memory Unit indicated by the two most significant bits of *raddr*. It is important to note that the *select* signal for this MUX is not directly connected to *raddr*. Instead, it passes through a D flip-flop (**DFF**). This design choice ensures that fluctuations in the *raddr* signal between clock cycles do not inadvertently alter the Memory Unit connected to the module's *dout*, thereby maintaining signal integrity and preventing glitches in the output.

▲ Figure 3.4: Multi-Bank Memory

Figure 3.4 illustrates the **Multi-Bank Memory** module, which represents the final implementation of our design. The structure of this module bears a strong resemblance to that of the Memory Bank connected to four Memory Units, as previously discussed. The primary distinction lies in the further extension of *waddr* and *raddr* signals by an additional two bits. Given the structural similarities, an additional description of the design is deemed unnecessary.

# Advanced Q4. Round-Robin FIFO Arbiter

# Advanced Q5. Exhaustive testbench design

# FPGA: 4‑bit Parameterized Ping‑Pong Counter

# Testbenches

## A. 4-bit Ping-Pong Counter

In this testbench, we initially use *rst\_n* to reset all values. We then run 32 clock cycles. Subsequently, we test the enable signal by turning it on for one clock cycle and off for one clock cycle, followed by turning it on for two clock cycles and off for two clock cycles. All operations function normally.

## B. First-In First Out (FIFO) Queue

The testbench for this Advanced Q4 has been constructed in strict accordance with the module diagrams provided in the assignment presentation. As such, further elaboration on its structure is not deemed necessary.

## C. Multi-Bank Memory

Given that it is not feasible to comprehensively test all possible values for this problem, we generate and test edge cases. These include scenarios where both *ren* and *wen* signals are set to one, cases where only one of them is set to one, and instances where neither is set to one.

## D. Round-Robin FIFO Arbiter

The testbench for this Advanced Q4 has been constructed in strict accordance with the module diagrams provided in the assignment presentation. As such, further elaboration on its structure is not deemed necessary.

## E. 4-bit Parameterized Ping-Pong Counter

# What have we learned from Lab 3?

# Contributions

謝佳晉：

wrote Verilog modules: Q1, Q2, Q4

wrote testbenches: Q1, Q2, Q4, Q5

performed simulation: Q1, Q2, Q4, Q5

drew diagram: basic Q1, Q2

wrote report: basic Q3, Q5, tbQ1, tbQ2, tbQ4

made FPGA demonstration

范升維：

wrote Verilog modules: Q1, Q2, Q3, Q4

wrote testbench: Q3

performed simulation: Q1, Q2, Q3, Q4 simulation on Vivado

drew diagram: ALL (basic Q1, Q1, Q2, Q3, Q4)

wrote report: basic Q1, basic Q3, Q1, Q2, Q3, Q4, tbQ3, what we learned

organized whole report

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | b-Q1 | b-Q3 | Q1 | Q2 | Q3 | Q4 | Q5 | FPGA | What we learned |
| wrote Verilog modules |  |  |  |  |  |  |  |  |  |
| wrote testbenches |  |  |  |  |  |  |  |  |  |
| performed simulation |  |  |  |  |  |  |  |  |  |
| drew diagram |  |  |  |  |  |  |  |  |  |
| wrote report |  |  |  |  |  |  |  |  |  |
| wrote report tb |  |  |  |  |  |  |  |  |  |

|  |
| --- |
| Both |
| 謝佳晉 |
| 范升維 |